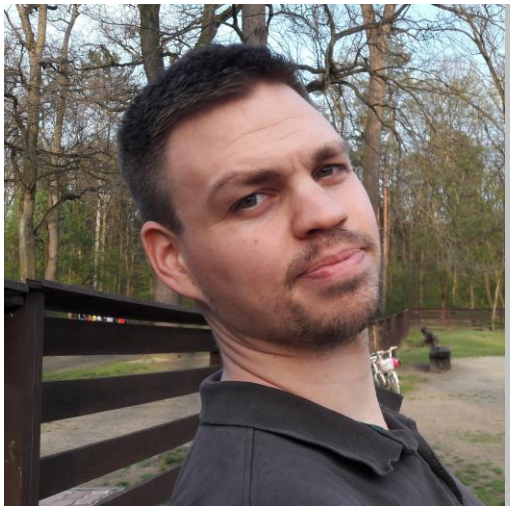


Dr. Hegedűs Péter, Dr. Ferenc Rudolf

Nagyméretű adatbázisok

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

NoSQL Hadoop felett - Apache HBase

Összefoglalás

Ez az olvasólecke gyakorlati ismereteket nyújt az Apache HBase rendszer használatáról. Bemutatjuk, hogyan kell önálló és elosztott módban telepíteni a HBase szervert, illetve hogyan tudjuk virtualizált környezetben elindítani azt. Részletes példákon keresztül bemutatjuk a hbase shell parancssori alkalmazás használatát, az egyes adat modell és adat manipulációs HBase utasításokkal együtt. Betekintést kaphat az olvasó abba is, hogyan lehet Java kódból natív módon elérni a HBase szervert és hozzáférni a tárolt adatokhoz.
A lecke fejezetei:

- 1. fejezet: **Apache HBase telepítése, docker stack indítása és a parancssori kliens használata (olvasó)**
- 2. fejezet: **Apache HBase programozása a natív Java API-n keresztül (olvasó)**

Téma típusa: **gyakorlati**
Olvasási idő: **50 perc**

📖 1. fejezet

Apache HBase indítása, parancssori kliens használata

HBase telepítése és docker stack indítása

Ahogy a kapcsolódó előadás olvasóleckéiben (`7e_BigData-nosql-over-hadoop-SPOC`) már láttuk, az Apache HBase [1] egy NoSQL adatbázis, ami a HDFS-t használja az adatok elosztott tárolásához. A HBase telepítése a megfelelő bináris csomag letöltéséből, kicsomagolásából, valamint a környezeti változók és konfigurációs állományok beállításából áll. Részletek a hivatalos dokumentációban [2] olvashatók. Mi a lokális gépre történő telepítés helyett egy előre előkészített docker container stack-et fogunk használni, mert a HBase használatához egy Hadoop klaszterre is szükség van. A következőkben bemutatott stack az összes szükséges docker image-t elindítja, ami szükséges a HBase azonnali használatához. A következő példák tetszőleges gépen futtathatók, ahol telepítve van a Dokcer környezet, valamint a Git verziókövető kliens.

A HBase stack indításához először töltsük le a docker leírókat és a teljes stack konfigurációt tartalmazó git repository-t a következő parancs segítségével:

```
$ git clone https://github.com/big-data-europe/docker-hbase.git
```

A letöltött `docker-hbase` mappa két konfigurációs állományt is tartalmaz: `docker-compose-standalone.yml` és `docker-compose-distributed-local.yml`. Mi a `docker-compose-standalone.yml`-t fogjuk használni, amely ún. standalone módban indítja a HBase-t.

Standalone konfiguráció
Soha ne használjuk a HBase standalone módját éles környezetben! Ez csak és kizárólag a fejlesztést és tesztelést könnyítő konfigurációs lehetőség. Az éles környezetben a HBase-t mindig elosztott módon, klaszterben futtassuk, hogy kihasználhassuk a hatalmas adatok párhuzamos feldolgozásából adódó előnyöket!

A fent említett standalone konfiguráció az alábbi stack-et definiálja:

- `namenode` - a Hadoop klaszter NameNode szervere
- `datanode` - egy darab Hadoop DataNode
- `resourcemanager` - erőforrás kezelésért felelős node (Apache Yarn)
- `nodemanager1` - Hadoop Node Manager csomópont
- `historyserver` - Hadoop history server node
- `hbase` - az Apache HBase server standalone módban

Az elosztott konfiguráció stack-je a HBase standalone szervere helyett három másik container-t indít:

- `zoo` - ZooKeeper szolgáltatás a konfigurációk és elosztott szinkronizálás megvalósításához
- `hbase-master` - a HBase ~~master~~ primary node-ja
- `hbase-region` - a HBase egy RegionServer-e

A standalone stack indításához lépjünk be a `docker-hbase` mappába, és adjuk ki a következő docker parancsot:

```
$ docker-compose -f docker-compose-standalone.yml up -d
```

A stack sikeres indítása után a következő paranccsal ellenőrizhetjük, hogy minden container sikeresen elindult:

```
$ docker ps
```

Ütköző container nevek
Ha korábban egy másik docker stack-et is indítottunk, ahol bizonyos conainer-eknek ugyanaz volt a neve (pl. *namenode*, *datanode* elég gyakori), akkor a docker panaszkodhat, hogy ilyen nevű container már létezik. Az összes ilyen container-t töröljünk (az azonosítójuk is kiíródik) a következő parancssal: *docker rm {container_id}*!

HBase parancssori kliens használata

A HBase azonnali használatához a HBase parancssori kliens eszközt (`hbase shell`) [3] használhatjuk. Ehhez lépjünk be a `hbase` szerver container-be, és adjuk ki az alábbi parancsot:

```
$ docker exec -it hbase bash
root@87fb9ec031da:/# hbase shell
2020-08-24 08:58:30,239 WARN  [main] util.NativeCodeLoader: Unable to load
native-hadoop library for your platform... using builtin-java classes where
applicable
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017

hbase(main):001:0>
```

Hozzunk létre egy táblát `hr_table` néven, két oszlop családdal: `personal` és `salary`. Az első a HR nyilvántartásban szereplő személyes adatait írja le, a másik pedig a fizetési adatokat. A bejegyzések hiányosak is lehetnek, azaz nem kell minden sorban minden adatot kitölteni, valamint lehetnek olyan értékek, ami csak bizonyos soroknál jelennek meg. Lássuk a tábla létrehozó utasítást!

```
hbase(main):006:0> create 'hr_table', 'personal', 'salary'
0 row(s) in 1.2390 seconds

=> Hbase::Table - hr_table
```

A létrehozott tábla leírását a `describe` parancssal tudjuk lekérdni:

```
hbase(main):007:0> describe 'hr_table'
Table hr_table is ENABLED
hr_table
COLUMN FAMILIES DESCRIPTION
{NAME => 'personal', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY =>
'false', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL =>
'FOREVER', COMPRESSION => 'NO
NE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536',
REPLICATION_SCOPE => '0'}
{NAME => 'salary', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false',
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
COMPRESSION => 'NONE
', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536',
REPLICATION_SCOPE => '0'}
2 row(s) in 0.0350 seconds
```

Két oszlop család jött létre, mindegyikhez láthatjuk azok tulajdonságait is (tömörítés, tárolás helye, stb.). Ezután szúrjunk be adatokat a `put` parancs segítségével:

```
hbase(main):003:0> put 'hr_table', '123456AB', 'personal:name', 'Kovacs Janos'
0 row(s) in 0.0560 seconds
```

Egy sort adtunk hozzá a táblához, a sor azonosítója az `123456AB`, egy oszlop érték a név, ami a `personal` családba tartozik, és a `name` qualifier-t használja. Adjunk hozzá még néhány adatot a táblához!

HBase shell
A HBase shell nem támogatja egyszerre több cella beszúrását, így azokat egyesével kell hozzáadnunk. Azonban, ha az egyszerre történő hozzáadást szeretnénk szimulálni, használhatjuk implicit módon ugyanazt a timestamp-et.

```
hbase(main):001:0> put 'hr_table', '123456AB', 'personal:city', 'Budapest'
0 row(s) in 0.2040 seconds

hbase(main):002:0> put 'hr_table', '123456AB', 'salary:amount', '654321'
0 row(s) in 0.0110 seconds

hbase(main):003:0> put 'hr_table', '123456AB', 'salary:position', 'CEO'
0 row(s) in 0.0130 seconds

hbase(main):004:0> put 'hr_table', '987654XY', 'personal:name', 'Mekk Elek'
0 row(s) in 0.0050 seconds

hbase(main):005:0> put 'hr_table', '987654XY', 'salary:position', 'CTO'
0 row(s) in 0.0080 seconds

hbase(main):006:0> put 'hr_table', '987654XY', 'salary:expires', '2020-12-31'
0 row(s) in 0.0080 seconds
```

Látszik, hogy nem ugyanazokat az oszlopokat töltöttük fel az egyes sorokhoz, noha van átfedés. Miután feltöltöttük adatokkal a táblát, nézzük a lekérdező műveleteket:

```
hbase(main):007:0> scan 'hr_table'
ROW                                           COLUMN+CELL
 123456AB                                     column=personal:city,
timestamp=1598262559466, value=Budapest
 123456AB                                     column=personal:name,
timestamp=1598262096119, value=Kovacs Janos
 123456AB                                     column=salary:amount,
timestamp=1598262588003, value=654321
 123456AB                                     column=salary:position,
timestamp=1598262596847, value=CEO
 987654XY                                     column=personal:name,
timestamp=1598262624631, value=Mekk Elek
 987654XY                                     column=salary:expires,
timestamp=1598262665668, value=2020-12-31
 987654XY                                     column=salary:position,
timestamp=1598262636861, value=CTO
2 row(s) in 0.0360 seconds

hbase(main):010:0> scan 'hr_table', {COLUMNS => 'personal:name'}
ROW                                           COLUMN+CELL
 123456AB                                     column=personal:name,
timestamp=1598262096119, value=Kovacs Janos
 987654XY                                     column=personal:name,
timestamp=1598262624631, value=Mekk Elek
2 row(s) in 0.0170 seconds

hbase(main):011:0> scan 'hr_table', {COLUMNS => 'salary:expires'}
ROW                                           COLUMN+CELL
```

```
 987654XY                                    column=salary:expires,
timestamp=1598262665668, value=2020-12-31
1 row(s) in 0.0120 seconds

hbase(main):012:0> scan 'hr_table', {COLUMNS => 'salary'}
ROW                                          COLUMN+CELL
 123456AB                                    column=salary:amount,
timestamp=1598262588003, value=654321
 123456AB                                    column=salary:position,
timestamp=1598262596847, value=CEO
 987654XY                                    column=salary:expires,
timestamp=1598262665668, value=2020-12-31
 987654XY                                    column=salary:position,
timestamp=1598262636861, value=CTO
2 row(s) in 0.0090 seconds

hbase(main):014:0> get 'hr_table', '123456AB'
COLUMN                                       CELL
 personal:city                               timestamp=1598262559466,
value=Budapest
 personal:name                               timestamp=1598262096119,
value=Kovacs Janos
 salary:amount                               timestamp=1598262588003,
value=654321
 salary:position                             timestamp=1598262596847,
value=CEO
4 row(s) in 0.0280 seconds

hbase(main):019:0> get 'hr_table', '123456AB', 'salary:amount'
COLUMN                                       CELL
 salary:amount                               timestamp=1598262588003,
value=654321
1 row(s) in 0.0030 seconds

hbase(main):024:0> get 'hr_table', '123456AB', {TIMERANGE => [1598262096119,
1598262559467]}
COLUMN                                       CELL
 personal:city                               timestamp=1598262559466,
value=Budapest
 personal:name                               timestamp=1598262096119,
value=Kovacs Janos
2 row(s) in 0.0090 seconds
```

A `scan` művelettel a teljes tábla tartalmat kaphatjuk vissza, természetesen szűréseket adhatunk a lekérdezéshez. A `get` utasítás egy adott sorhoz ad vissza értékeket (azt, hogy mit, természetesen itt is szűrhető). Adatok törlésére pedig a `delete` művelet ad lehetőséget (a `deleteall` egy sor összes oszlop értékét törli):

```
hbase(main):028:0> delete 'hr_table', '123456AB', 'salary:position'
0 row(s) in 0.0040 seconds

hbase(main):029:0> scan 'hr_table'
ROW                                          COLUMN+CELL
 123456AB                                    column=personal:city,
timestamp=1598262559466, value=Budapest
```

```
 123456AB                                          column=personal:name,
timestamp=1598262096119, value=Kovacs Janos
 123456AB                                          column=salary:amount,
timestamp=1598262588003, value=654321
 987654XY                                          column=personal:name,
timestamp=1598262624631, value=Mekk Elek
 987654XY                                          column=salary:expires,
timestamp=1598262665668, value=2020-12-31
 987654XY                                          column=salary:position,
timestamp=1598262636861, value=CTO
2 row(s) in 0.0260 seconds

hbase(main):031:0> deleteall 'hr_table', '987654XY'
0 row(s) in 0.0030 seconds

hbase(main):032:0> scan 'hr_table'
ROW                                               COLUMN+CELL
 123456AB                                          column=personal:city,
timestamp=1598262559466, value=Budapest
 123456AB                                          column=personal:name,
timestamp=1598262096119, value=Kovacs Janos
 123456AB                                          column=salary:amount,
timestamp=1598262588003, value=654321
1 row(s) in 0.0140 seconds
```

📖 2. fejezet

Apache HBase programozása Java-ban

Ebben a fejezetben bemutatjuk, hogyan lehet a HBase-ben tárolt adatokhoz hozzáférni, azokat manipulálni programkódból. Mivel a HBase rendelkezik Thrift [4] interfész leíró szolgáltatással, azon keresztül gyakorlatilag tetszőleges nyelvű kliensből tudunk csatlakozni hozzá. Az alábbi példakód Java nyelven készült, ami viszont a HBase által nyújtott natív Java API-t használja. A példakód (`code/8g_BigData-nosql-over-hadoop-SPOC/HBaseNativeClient.java`) először újabb adatokat szúr be a `hr_table` táblába, majd lekérdezi és kiíratja azokat. A kód váza így néz ki:

```java
package org.example;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;

/**
 * A sample HBase client that uses the native HBase API
 *
 */
public class HBaseNativeClient
{
    ...

    public static void main( String[] args ) throws IOException {
        Configuration config = HBaseConfiguration.create();
```

```
        Connection connection = ConnectionFactory.createConnection(config);
        HBaseAdmin.available(config);

        insertNewData(connection);
        getData((connection));
        scanData(connection);
    }
}
```

A kapcsolódás inicializálása nagyon egyszerű, egy `HBaseConfiguration` objektum alapján egy `Connection`-t kell létrehozni, aminek a segítségével aztán a megfelelő adatmanipuláló műveleteket el tudjuk végezni. Mivel az alap beállításokkal dolgozunk, nem kell módosítani semmin, egyébként a konfigurációnak tudnánk property-ket beállítani. A projekthez a `hbase-*.jar` függőségek kellenek (lásd `lib` könyvtár). Az új adat beszúrását végző kódrészlet a következő:

```
public static void insertNewData(Connection connection) throws IOException {
    Table table = connection.getTable(TableName.valueOf("hr_table"));

    // instantiate Put class
    Put p = new Put(Bytes.toBytes("123456AB"));

    // update value using add() method
    p.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("gender"),
Bytes.toBytes("male"));
    p.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("birthYear"),
Bytes.toBytes("1992"));

    // save the put Instance to the HTable.
    table.put(p);
    Put p2 = new Put(Bytes.toBytes("565656HH"));
    p2.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("name"),
Bytes.toBytes("John Teller"));
    p2.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("city"),
Bytes.toBytes("Charming"));

    // save the put Instance to the HTable.
    table.put(p2);
    System.out.println("data updated successfully");

    table.close();
}
```

A `Connection` segítségével lekérhetünk egy `Table` példányt, amin aztán `put()´` és egyéb műveletet hajthatunk végre. Minden műveletnek megvan a megfelelő Java osztálya. Létrehozunk egy `Put` objektumot meglévő sor kulccsal és egy teljesen újat is. Az `addColumn()` segítségével hozzáadjuk az osztályt, qualifier-t és értéket, majd elvégezzük a műveletet és lezárjuk a táblázatot. A többi metódus hasonlóan épül fel, a kódjuk a következő:

```
public static void getData(Connection connection) throws IOException {
    Table table = connection.getTable(TableName.valueOf("hr_table"));

    // instantiate Get class
    Get g = new Get(Bytes.toBytes("565656HH"));
```

```
    g.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("city"));

    // get the Result object
    Result result = table.get(g);

 System.out.println(Bytes.toString(result.getValue(Bytes.toBytes("personal"),Byt
es.toBytes("city"))));
    table.close();
}

public static void scanData(Connection connection) throws IOException {
    Table table = connection.getTable(TableName.valueOf("hr_table"));

    // instantiate the Scan class
    Scan scan = new Scan();

    // scan the columns
    scan.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("name"));
    scan.addColumn(Bytes.toBytes("personal"), Bytes.toBytes("city"));

    // get the ResultScanner
    ResultScanner scanner = table.getScanner(scan);
    for (Result result = scanner.next(); result != null; result=scanner.next())
{
        System.out.println("Found row : " + Bytes.toString(result.getRow()));
        System.out.println("\tName : " +
Bytes.toString(result.getValue(Bytes.toBytes("personal"),
Bytes.toBytes("name"))));
        System.out.println("\tCity : " +
Bytes.toString(result.getValue(Bytes.toBytes("personal"),
Bytes.toBytes("city"))));
    }
    scanner.close();
    table.close();
}
```

A program fordítása és futtatása előtt néhány változtatást kell tennünk a docker stack-en. Módosítsuk a `docker-compose-distributed-local.yml` fájlt, adjuk hozzá a `16000:16000`-es port átirányítást a `hbase-master` container-hez és a `16020:16020`-at a `hbase-region` container-hez. Állítsuk le a `standalone` konfigurációt, majd indítsuk el a `distributed local`-t:

```
$ docker-compose docker-compose-standalone.yml stop
$ docker-compose docker-compose-distributed-local.yml up -d
```

Ezen felül a lokális operációs rendszerünk `hosts` fájljába (`/etc/hosts` vagy `C:\Windows\System32\Drivers\etc\hosts`) be kell tennünk az alábbi két bejegyzést:

```
127.0.0.1 hbase-master
127.0.0.1 hbase-region
```

Ezek után fordítsuk le a Java programot és futtassuk az alábbi utasításokkal:

```
$ javac -cp lib/* org/example/HBaseNativeClient.java
$ java -cp .:lib/* org.example.HBaseNativeClient
Charming
Found row : 123456AB
    Name : Kovacs Janos
    City : Budapest
Found row : 565656HH
    Name : John Teller
    City : Charming
Found row : 987654XY
    Name : Mekk Elek
    City : null
```

További feladatok

1. Vigyük fel a `personal_entries.json`, `billing_entries.json`, és `sales_entries.csv` (a `2g_BigData-data-transform-SPOC` leckéből) fájlok tartalmát egy HBase adattáblába a parancssori kliens segítségével! ⭐
2. Írjunk egy Java programot, ami az előbb felvitt táblából lekéri az összes 1970 után született személy adatait, és kiírja azt! ⭐
3. Valósítsuk meg 2. fejezetben bemutatott Java program funkcionalitását Python nyelven! Használjunk külső könyvtárat (pl. HappyBase [5]) a HBase eléréséhez! ⭐⭐
4. Hozzunk létre egy olyan Flume adatfolyamot, amely a `netcat` adat forrásból érkező adatokat közvetlenül egy HBase táblába menti! Lásd HBase sink [6, 7]! ⭐⭐⭐
5. Integráljuk össze az Apache Hive és HBase rendszereket! Azaz tegyük elérhetővé a HiveQL lekérdezéseket egy HBase táblában tárolt adatok lekéréséhez! ⭐⭐⭐

Referenciák

[1] https://hbase.apache.org/

[2] https://hbase.apache.org/book.html

[3] https://hbase.apache.org/book.html#shell

[4] https://thrift.apache.org/

[5] https://happybase.readthedocs.io/en/latest/

[6] https://flume.apache.org/FlumeUserGuide.html

[7] https://blogs.apache.org/flume/entry/streaming_data_into_apache_hbase